CSc 174 Database Management Systems

13. Concurrency Control

Ying Jin
Computer Science Department
California state University, Sacramento

Concurrency Control

- Transactions can execute concurrently.
- Problems when allow concurrent execution without control
  - The Lost Update Problem.

    This occurs when
    - two transactions that access the same database items and
    - have their operations interleaved in a way that makes the value of some database item incorrect.

The Lost Update Problem - Example

(a)

Time

T_1	T_2
read_item(X); X:=X-N;	
	read_item(X); X:=X+M;
write_item(X); read_item(Y);	
	write_item(X);
Y:=Y+N; write_item(Y);	

Item X has an incorrect value because its update by T_1 is "lost" (overwritten)

Dirty Read problem

- The Temporary Update (or Dirty Read) Problem.

  This occurs when

  - one transaction updates a database item
  - and then the transaction fails for some reason.
  - The updated item is accessed by another transaction before it is changed back to its original value.

Dirty Read problem - Example

(b)

Time

T_1	T_2

read_item(X);
X:=X-N;
write_item(X);	

	read_item(X);
	X:=X+M;
	write_item(X);

read_item(Y);	

Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X.

Solve problems

- Problems occurs without control.
- Concurrency Control
  - Concepts: schedule, serializable
  - Concurrency control technique

Transaction Schedule

- A **schedule** (or **history**) S of n transactions T1, T2, ..., Tn

  It is an ordering of the operations of the transactions subject to the constraint that:
  - for each transaction Ti that participates in S
  - the operations of Ti in S must appear in the same order in which they occur in Ti.
- Note, however, that operations from other transactions Tj <u>can be interleaved</u> with the operations of Ti in S.

Conflict operations

- Two operations in a schedule **conflict** if they satisfy all three of the following conditions:
  (1)They belong to different transactions
  (2)They access the same item X
  (3) At least one of the operations is a write_item(X)

Equivalent schedule

- **Result equivalent**: Two schedules are called result equivalent if they produce the same final state of the database.
  - Two different schedules may accidentally produce the same final state.
- **Conflict equivalent**: Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

Serial Schedule

- A schedule S is **serial** if,
  - for every transaction T participating in the schedule,
  - all the operations of T are executed consecutively in the schedule.

  (Without interleaving - Only one transaction active at once)
- Otherwise, the schedule is called **nonserial schedule.**

Serializable Schedule

- **Conflict serializable**: A schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule S'.
- We call it "serializable" in the rest of this CSC 174 class.
- Being serializable is <u>not</u> the same as being serial

Property of Serializable schedule

- Why study Serializablity?
- Being serializable implies that the schedule is a correct schedule.
  - It will leave the database in a consistent state.
  - The interleaving is appropriate and will result in a state as if the transactions were serially executed, yet will achieve efficiency due to concurrent execution.

Testing for conflict serializability

Algorithm 17.1:

1. Looks at only read_Item (X) and write_Item (X) operations
2. Constructs a precedence graph (serialization graph) - a graph with directed edges
3. An edge is created from T_i to T_j if one of the operations in T_i appears before a conflicting operation in T_j (RW,WR,WW)
4. The schedule is serializable if and only if the precedence graph has no cycles.

- **Example 1**

Example 2

(a)

transaction T_1	transaction T_2	transaction T_3
read_item (X);	read_item (Z);	read_item (Y);
write_item (X);	read_item (Y);	read_item (Z);
read_item (Y);	write_item (Y);	write_item (Y);
write_item (Y);	read_item (X);	write_item (Z);
	write_item (X);	

Example 2 (cont.)

(b)

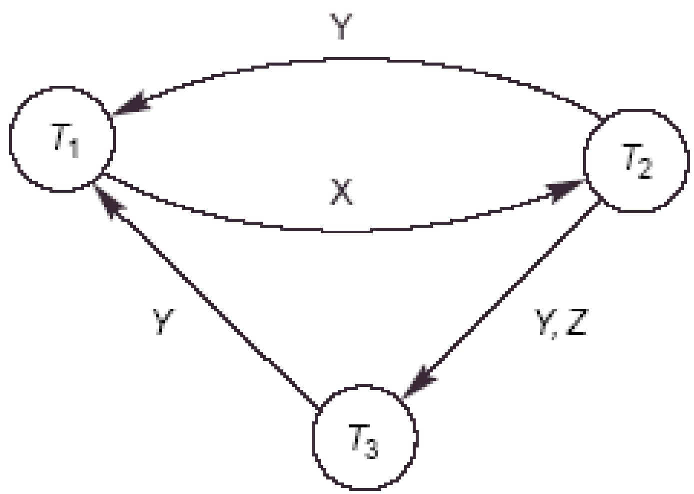
Time

transaction T_1	transaction T_2	transaction T_3
	read_item (Z);	
	read_item (Y);	
	write_item (Y);	
		read_item (Y);
		read_item (Z);
read_item (X);		
write_item (X);		
		write_item (Y);
		write_item (Z);
	read_item (X);	
read_item (Y);		
write_item (Y);		
	write_item (X);	

Schedule E

Example 2 (Cont.)

(d)



Equivalent serial schedules

None

Reason

cycle $X(T_1 \rightarrow T_2)$, $Y(T_2 \rightarrow T_1)$

cycle $X(T_1 \rightarrow T_2)$, $YZ(T_2 \rightarrow T_3)$, $Y(T_3 \rightarrow T_1)$

Exercise

(c)

transaction T_1	transaction T_2	transaction T_3
		read_item (Y);
		read_item (Z);
read_item (X);		
write_item (X);		
		write_item (Y);
		write_item (Z);
	read_item (Z);	
read_item (Y);		
write_item (Y);		
	read_item (Y);	
	write_item (Y);	
	read_item (X);	
	write_item (X);	

Time ↓

Schedule F

Use of Serializability

- In practice
  - Interleaving of operations from concurrent transactions is typically determined by the operating system scheduler
  - It's difficult to determine beforehand how the operations in a schedule will be interleaved.
- If transactions are executed at will and then resulting schedule is tested for serializability,

  → cancel the effect of non-serializable schedule.

  → impractical

Practical Approach

- Come up with methods (protocols) to ensure serializability.
- Current approach used in most DBMSs:

  Concurrency Control Techniques
  - Two-Phase locking
  - Timestamp Ordering

Lock

Locking is an operation which secures
(a) permission to Read or
(b) permission to Write a data item for a transaction.

Binary Locks:

- Lock (X). Data item X is locked in behalf of the requesting transaction.
- Unlock (X). Data item X is made available to all other transactions.

Rules for Binary locking

- Every transaction T must obey the following rules:
  - T must issue the operation lock(x) before any read(x) or write(x) operations are performed in T.
  - T must issue the operation unlock(x) after all read(x) and write(x) operations are completed in T.
  - T will not issue lock(x) operation if it already holds the lock on item x
  - T will not issue an unlock(x) operation unless it already holds the lock on item X.
- At most one transaction can hold the lock on a particular item.

Shared/Exclusive locks

- The above binary locking is too restrictive.
- We should allow several transactions to access the same item X if they all access x for reading purpose
- Three locking operations: read_lock(x), write_lock(x), and unlock(x).

  Shared lock: read_lock (X). More than one transaction can apply share lock on X for reading its value.

  Exclusive lock: write_lock(x). Only a single transaction exclusively holds the lock on the item.

Lock Compatibility

- T is requesting a lock.
- Lock: The lock manager issue a lock to T.
- Wait: T has to wait until the lock is issued by the lock manager.

	READ REQUEST	WRITE REQUEST
READ LOCK	LOCK	WAIT
WRITE LOCK	WAIT	WAIT

Well-formed transactions

- Database requires that all transactions should be well-formed.
- A transaction is well-formed if:
  - It must lock the data item in an appropriate mode before it reads or writes to it. (read: read_lock, write: write_lock)
  - If the data item is already locked by another transaction in an incompatible mode, then T must wait until all incompatible locks held by other transactions have been released.
  - It must not try to unlock a free data item.

Locking Example

T1	T2	Result
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2
write_lock (X);	Write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Locking Example (Cont)

T1	T2	Result
read_lock (Y);		X=50; Y=50
read_item (Y);		Well-formed.
		Serializable?
unlock (Y);		Lock guarantee Serializable?
	read_lock (X);	Need more rules/ protocols.
	read_item (X);	
	unlock (X);	
	write_lock (Y);	
	read_item (Y);	
	Y:=X+Y;	
	write_item (Y);	
	unlock (Y);	
write_lock (X);		
read_item (X);		
X:=X+Y;		
write_item (X);		
unlock (X);		

Two-Phase Locking Protocol (2PL)

- A transaction is said to follow the **two-phase locking protocol** if all locking operations (read_lock, write_lock) precede the *first* unlock operation.
- Transaction can be divided into two phases
  - Growing (first) phase: new locks on items can be acquired but none can be released
  - Shrinking(second) phase: existing locks can be released but no new locks can be acquired.

2PL (Cont.)

- If every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable.

Example 1 of 2PL

<u>T’1</u>

```
read_lock (Y);
read_item (Y);
write_lock (X);
unlock (Y);
read_item (X);
X:=X+Y;
write_item (X);
unlock (X);
```

Example 2 of 2PL

T'2

read_lock (X);
read_item (X);
Write_lock (Y);
unlock (X);
read_item (Y);
Y:=X+Y;
write_item (Y);
unlock (Y);

- Example 3: Interleaving of T3 and T4.

Deadlock

- **Deadlock** occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T′ in the set.

Deadlock Example

T'1	T'2
read_lock (Y); read_item (Y);	
	read_lock (X); read_item (X);
write_lock (X); (waits for X)	
	write_lock (Y); (waits for Y)

T1 and T2 did follow two-phase policy, but they are deadlock

Deadlock (T'1 and T'2)

Deadlock Prevention

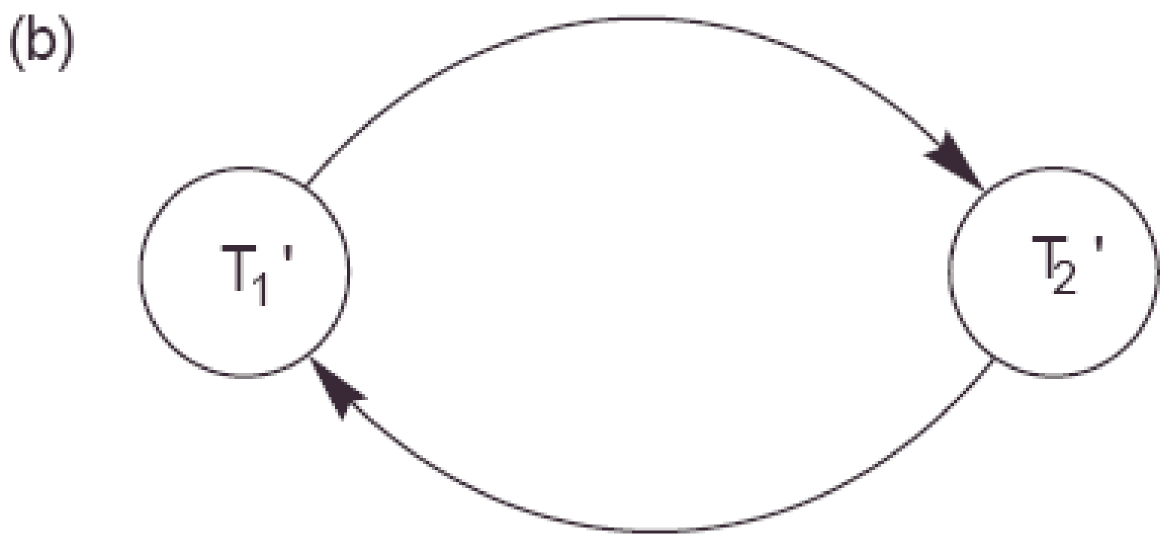
- Use deadlock prevention protocol to prevent deadlock.
  - Conservative two-phase locking:
    - Requires a transaction to lock all the items it accesses before the transaction begins execution.
    - If any of the items cannot be obtained, none of the items are locked.
    - Then the transaction will try again at a later time.
  - Not practical

Deadlock Detection

- DBMS checks if a state of deadlock actually exists.
- More practical approach
- Wait-for graph
  - One node for each transaction that is currently executing
  - Whenever Ti is waiting to lock an item X that is currently locked by a transaction Tj, a directed edge (Ti -> Tj) is created in the wait-for graph

Wait-for graph example

T1′ and T2′ of Page 33



Deadlock Detection (Cont.)

- Deadlock if an only if the wait-for graph has cycle
- DBMS needs to select a time to check for a deadlock.

Deadlock Detection (Cont.)

- When a deadlock occurs, DBMS will choose some transaction causing the deadlock to abort.
- Victim Selection algorithm:
  - Avoid selecting transactions that have been running for a long time and that have performed many update.
  - Select transactions that have not made many changes

Starvation

- Starvation occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
- This may occur if the waiting/aborting scheme for locked items is unfair.
  - Fair waiting scheme: first-come-first-served
  - Allow priority: Priority increasing during waiting.

Timeout

- Simple approach to deal with deadlock
- If a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it – regardless of whether a deadlock actually exists or not.
- Practical because of is low overhead and simplicity.

These slides are based on the textbook:

R. Elmasri and S. Navathe, *Fundamentals of Database System*, 7th Edition, Addison-Wesley.